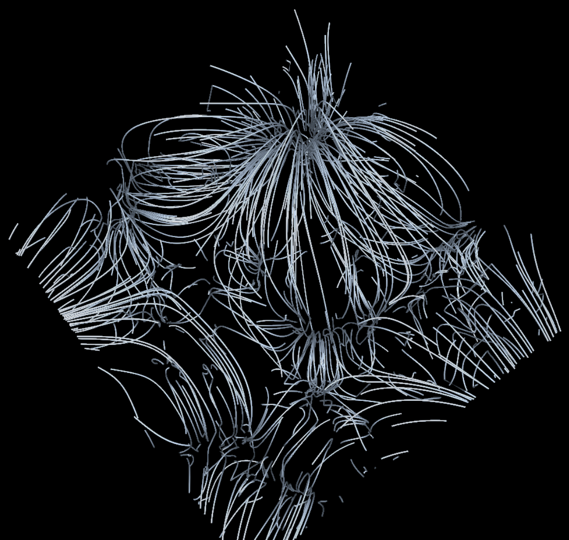
Data Movement Support for Analysis



Early stages of Rayleigh-Taylor Instability flow

Tom Peterka, Rob Ross *Argonne National Laboratory*

Abon Choudhuri, Teng-Yok Lee, Han-Wei Shen *The Ohio State University*

Wes Kendall *University of Tennessee, Knoxville*

Attila Gyulassy, Valerio Pascucci *University of Utah*

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division

CScADS Summer Workshop

7/26/11

Premises and Challenges

Premises

All analyses are different. The user is the expert.

The user already has a serial computational module for the analysis.

Parallelizing from scratch is daunting, steep learning curve, scalability not trivial.

Scalable data movement is the key.

A common set of operations can be identified and encoded in a library.

DIY helps the user parallelize their analysis algorithm with data movement tools.

Challenges

Data model: MPI datatypes (currently)

Execution model: in situ or postprocessing

Parallelism model: MPI message passing (currently)

Load balancing: Zoltan-based repartitioning (in the works)

DIY Structure

Features

Parallel input from storage

Parallel output to storage

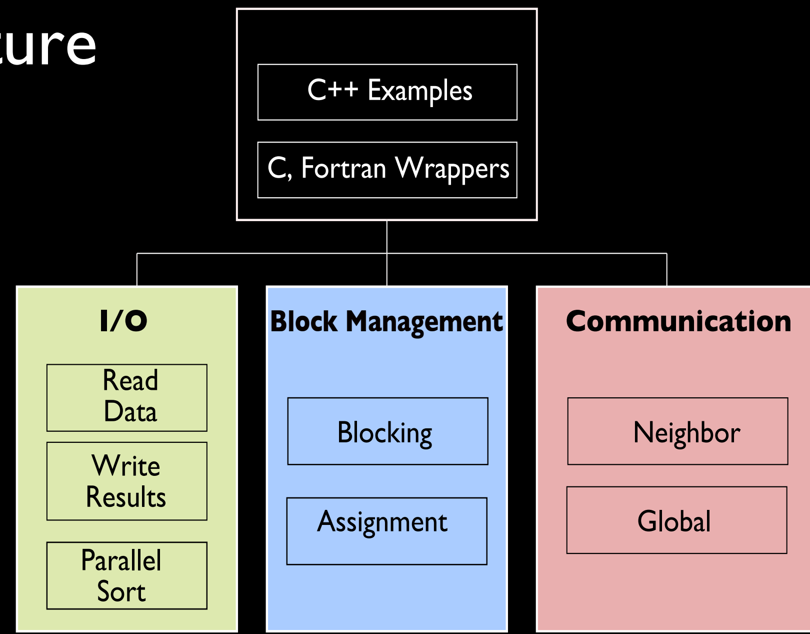
Domain decomposition

Network communication

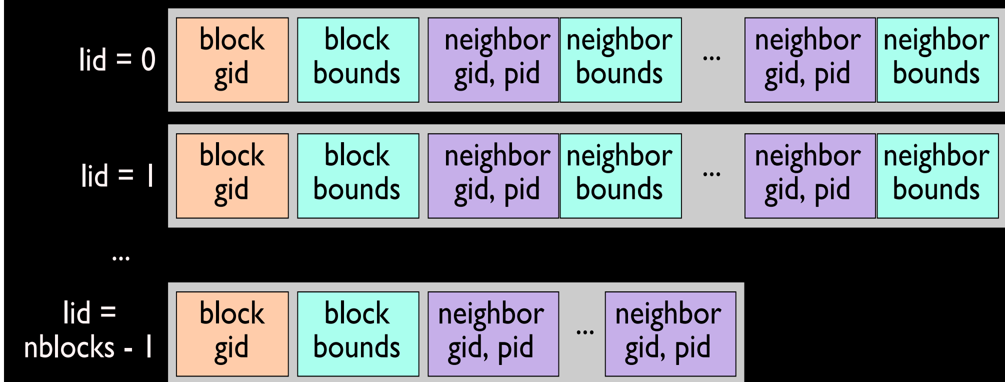
Library structure

Written in C++

Future wrappers for C and Fortran

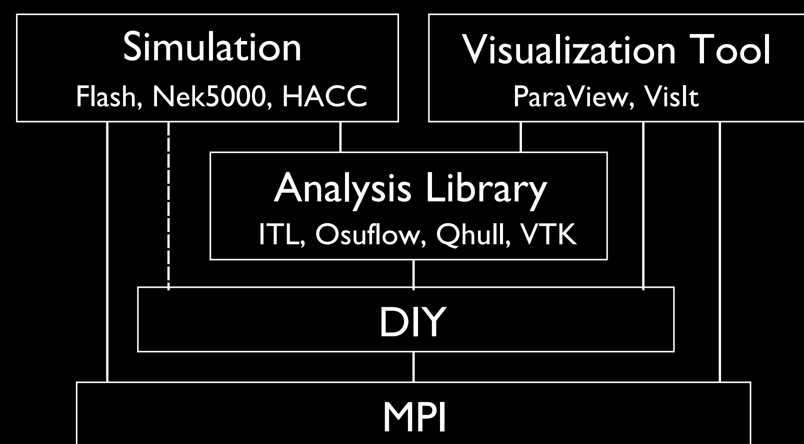


DIY library organization



Legend:
gid = global block identification
lid = local block identification
pid = process identification

DIY block list data structure



DIY usage

BIL: Input I/O (code contributed by Wes Kendall)

Application-level two-phase I/O

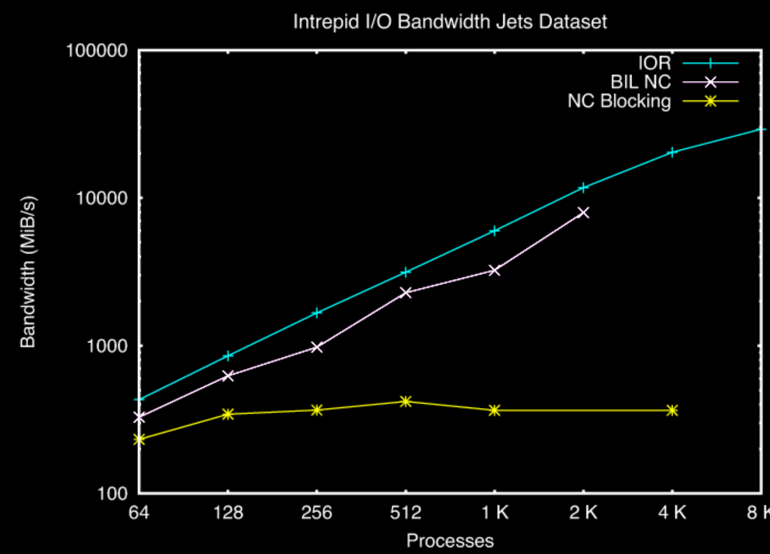
Reads raw, netCDF (current), HDF5 (future)

User posts requests

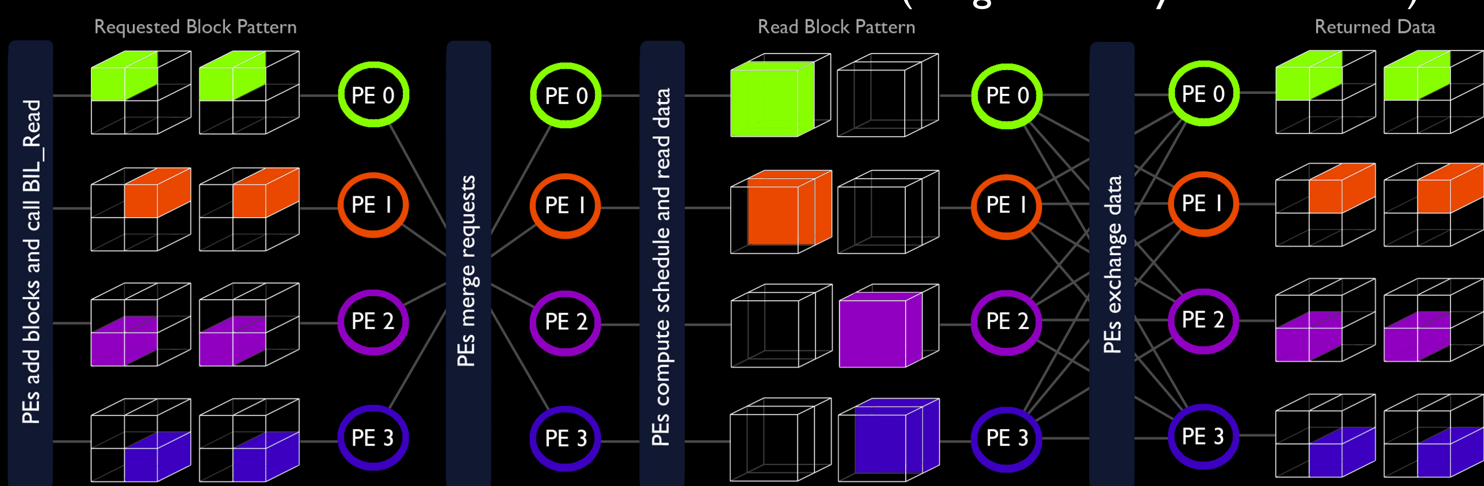
BIL sorts and aggregates them into large contiguous accesses

BIL redistributes data to processes after reading

Works on single and multi block/file domains.



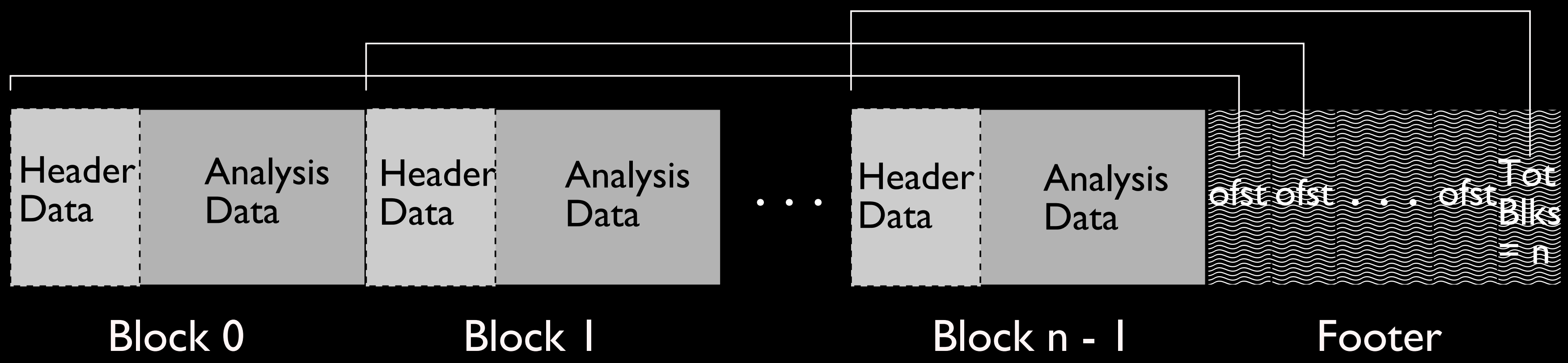
BIL performance 75% of IOR benchmark (image courtesy Wes Kendall)



BIL operations (image courtesy Wes Kendall)

Kendall et al., Towards a General I/O Layer for Parallel Visualization Applications, CG&A '11

Output I/O



Output file format

Features

Binary

General header/data blocks

Footer with indices

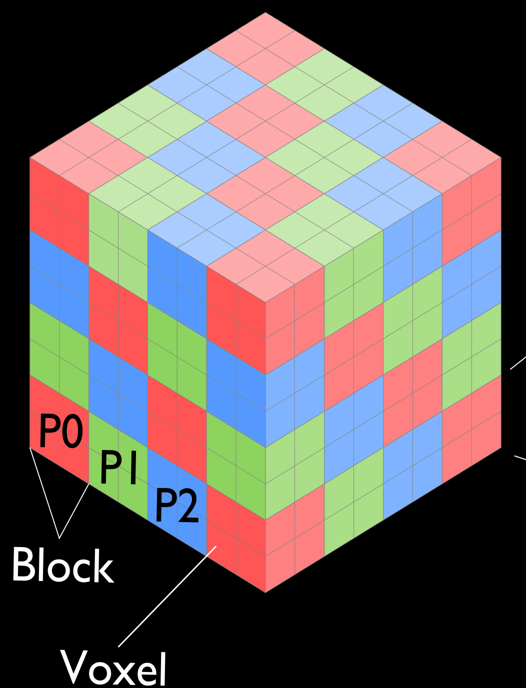
Application assigns semantic value, DIY deals only in MPI datatypes (pushes data model questions up to the application)

Written efficiently in parallel, at least on BG/P so far

Single file for now, time-varying output not done yet

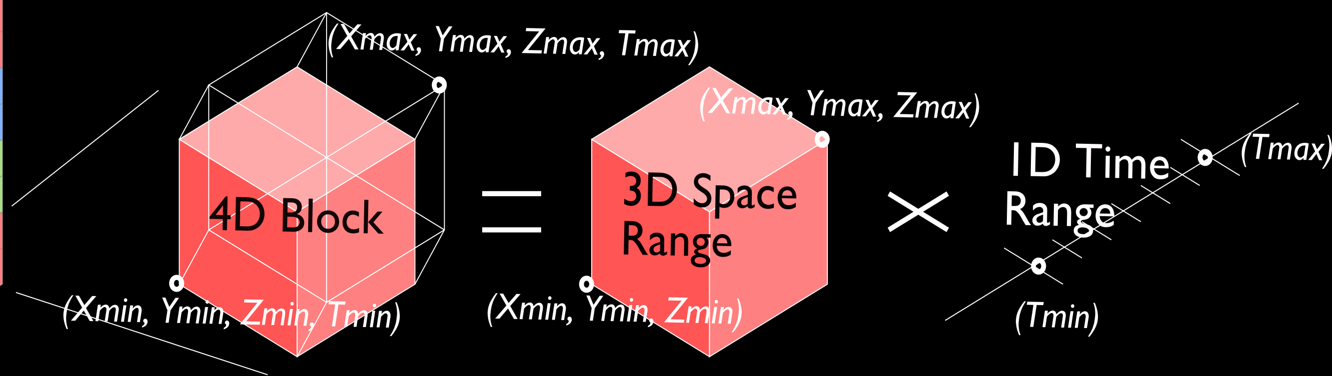
Blocking and Assignment

Process Assignment



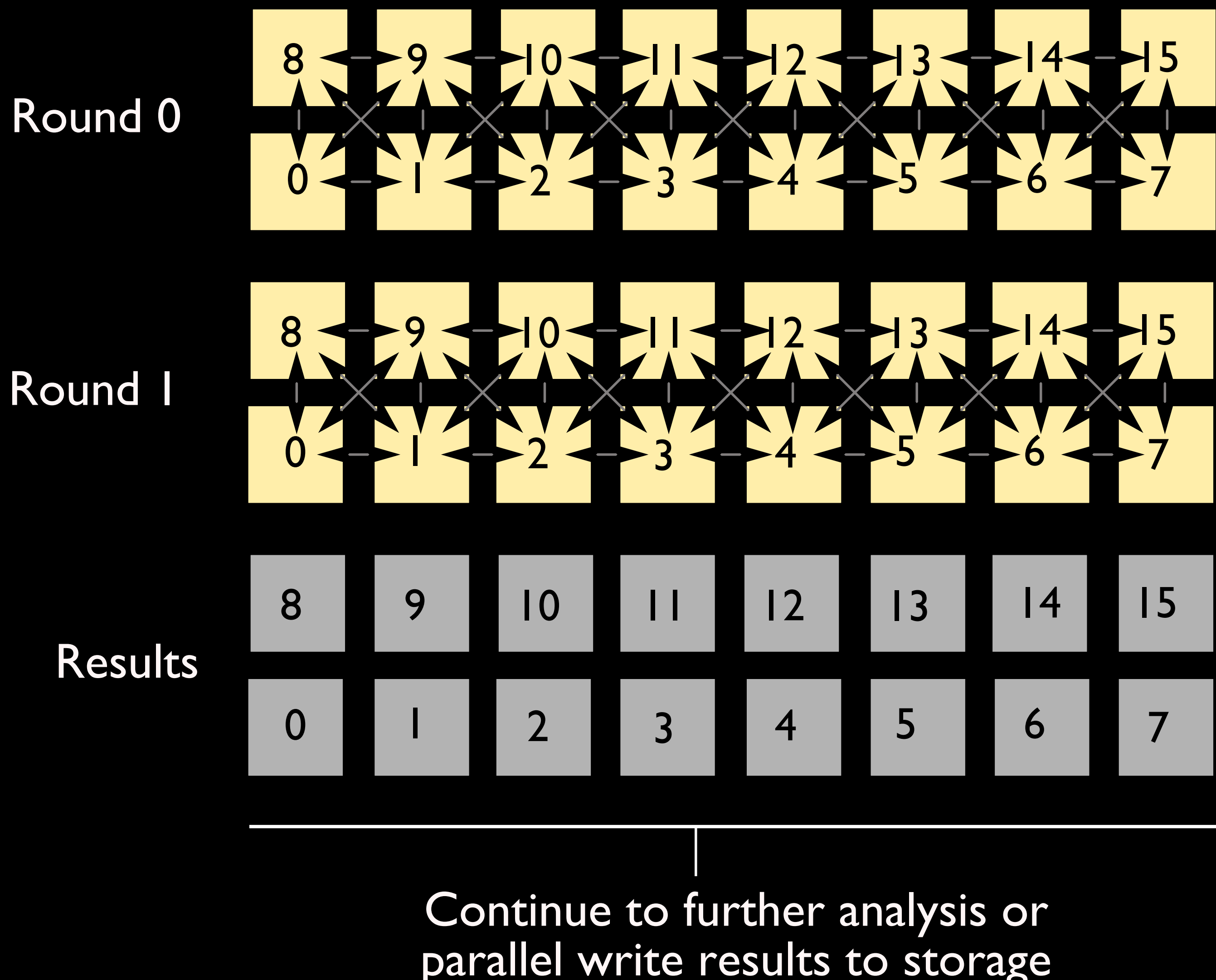
Example of multiblock assignment: 512 voxels decomposed into 64 blocks and assigned to 3 processes.

Blocking



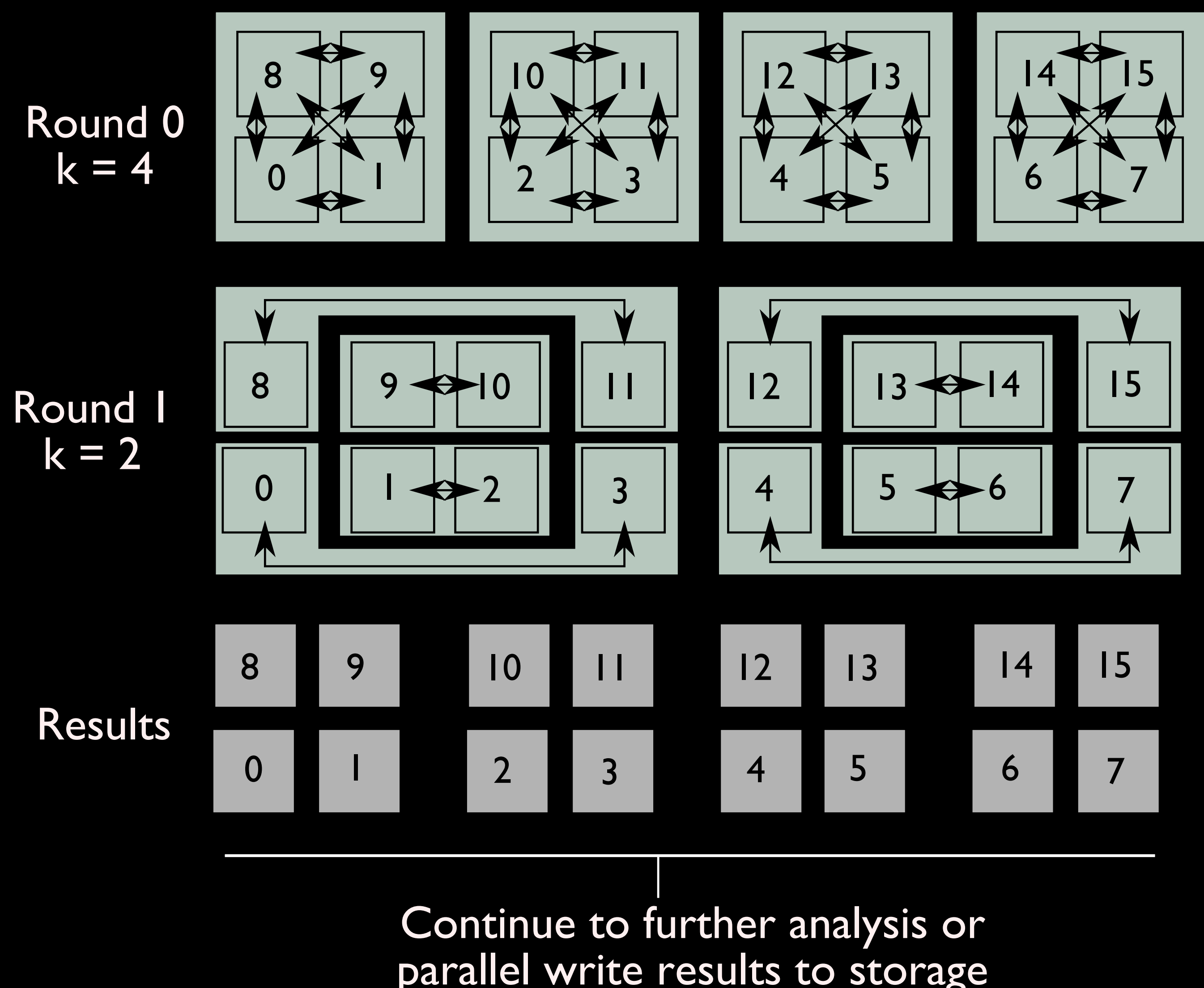
Hybrid 3D/4D time-space decomposition. Time-space is represented by 4D blocks that can also be decomposed such that time blocking is handled separately.

Communication Patterns: Nearest Neighbor



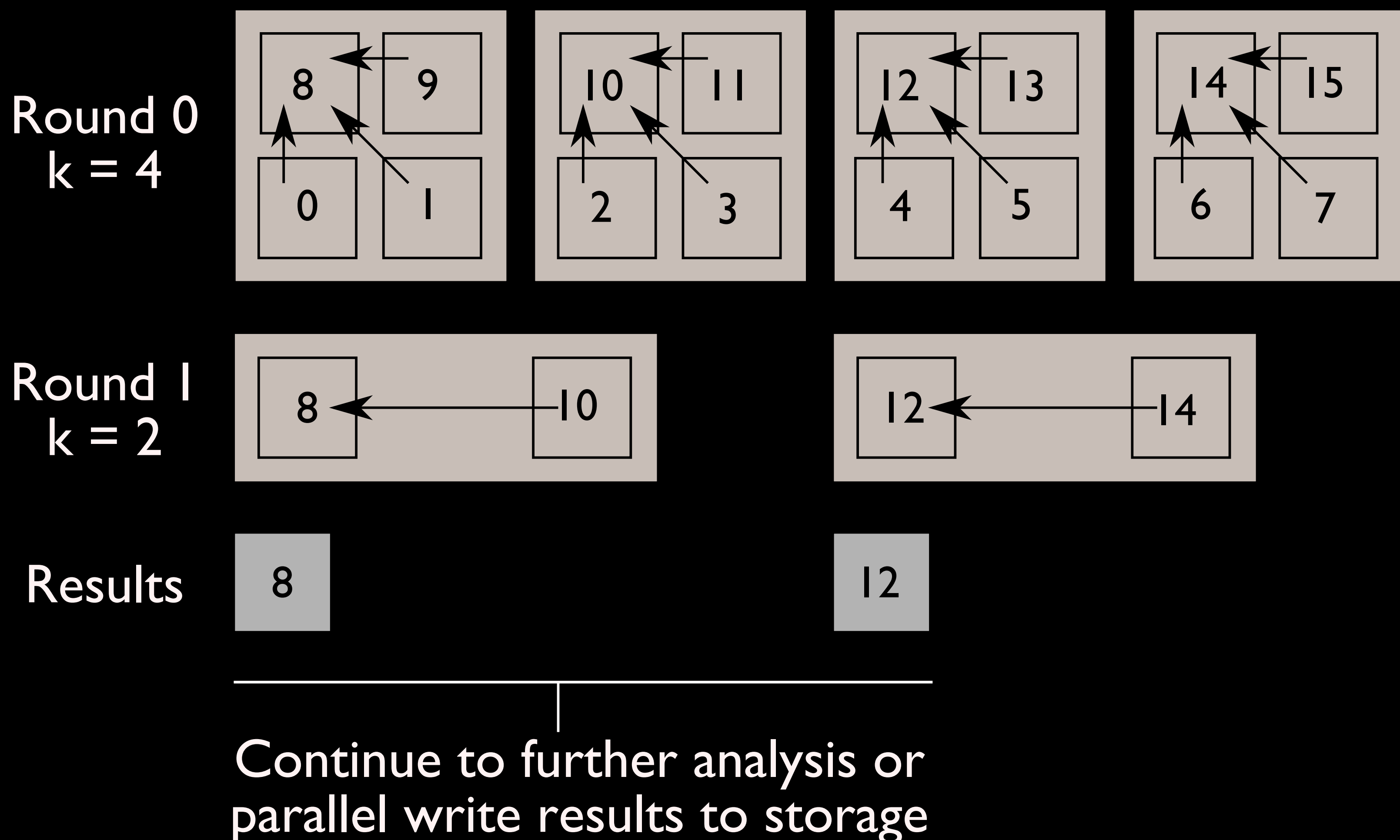
Peterka et al., A Study of Parallel Particle Tracing for Steady-State and Time-Varying Flow Fields, IPDPS '11

Communication Patterns: Swap-Based Global Reduction



Peterka et al., A Configurable Algorithm for Parallel Image-Compositing Applications, SC '09

Communication Patterns: Merge-Based Global Reduction



Peterka et al., Scalable Parallel Building Blocks for Custom Data Analysis, to appear in LDAV '11

Example API Use

```
// setup and domain decomposition
int dim = 3; // number of dimensions in the problem
int tot_blocks = 8; // total number of blocks
int64_t data_size[3] = {10, 10, 10}; // data size
int64_t given[3] = {0, 0, 0}; // constraints on blocking (none)
Assignment *assignment = new Assignment(tot_blocks, nblocks, maxblocks,
    MPI_COMM_WORLD);
Blocking *blocking = new Blocking(dim, tot_blocks, data_size, 0, 0, 0, given, assignment,
    MPI_COMM_WORLD);

// read data
 for (int i = 0; i < nblocks; i++) {
    blocking->BlockStartsSizes(i, min, size);
    int bil_data_size[3] = { data_size[2], data_size[1], data_size[0] };
    int bil_min[3] = { min[2], min[1], min[0] };
    int bil_size[3] = { size[2], size[1], size[0] };
    BIL_Add_block_raw(dim, bil_data_size, bil_min, bil_size, infile, MPI_INT, (void**)&(data[i]));
}
BIL_Read();
```

Example API Continued

```
// local analysis
…

// merge results
int rounds = 2; // two rounds of merging
int kvalues[2] = {4, 2}; // k-way merging, eg 4-way followed by 2-way merge
int nb_merged; // number of output merged blocks
Merge *merge = new Merge(MPI_COMM_WORLD);
nb_merged = merge->MergeBlocks((char**)merged_results, (int **)NULL, nblocks, (char**&)
results, rounds, &kvalues[0], io, assignment, &ComputeMerge, &CreateItem,&DeleteItem,
&CreateType);

// write results
IO *io = new IO(dim, tot_blocks, maxblocks, MPI_COMM_WORLD);
io->WriteAnaInit(outfile);
io->WriteAllAna((void **)merged_results, nb_merged, maxblocks, dtype);
io->WriteAnaFinalize();
```

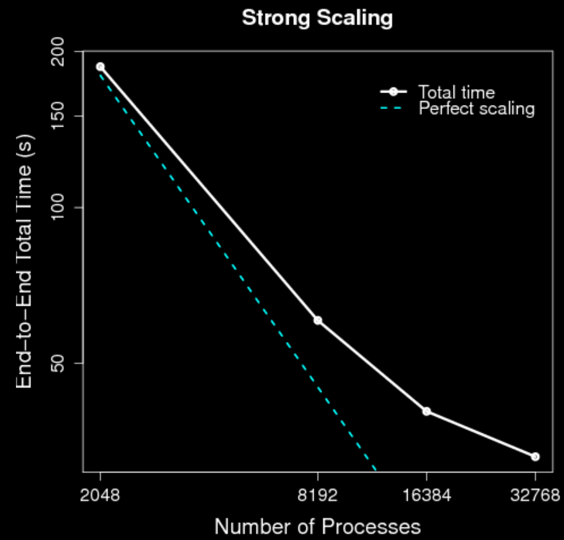
Applications and Results

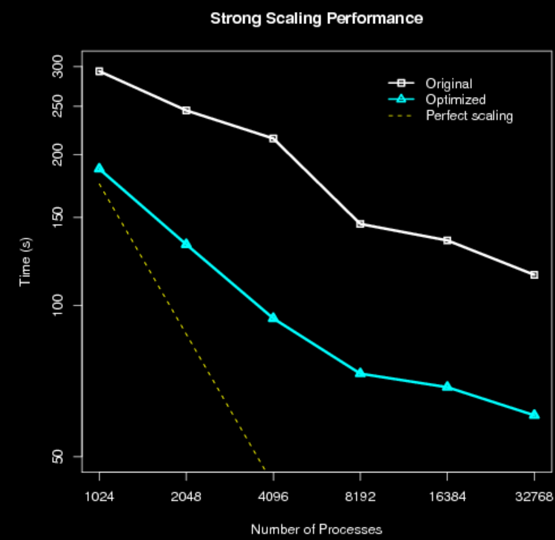
Particle tracing

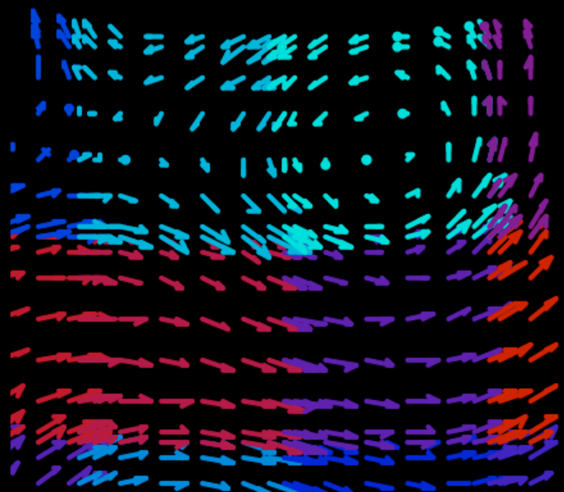
Morse-Smale complex

Information entropy

Feature detection

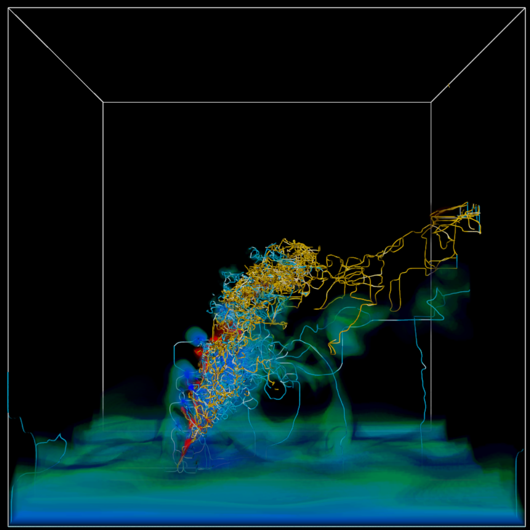




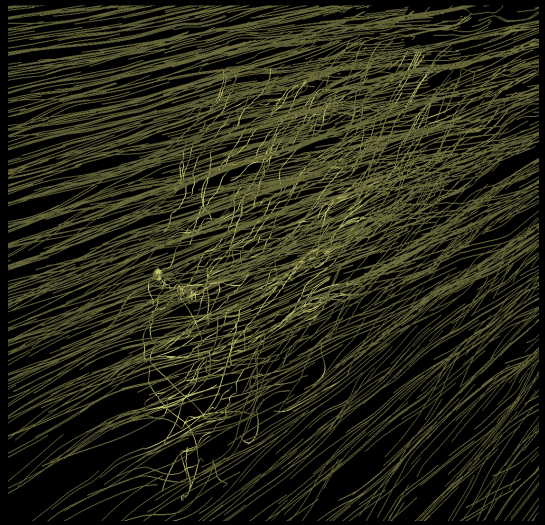


Information entropy

(image courtesy Teng-Yok Lee)



Topological analysis



Streamline and pathline tracing

Summary

Successes

Supports numerous, diverse analysis techniques.

  Flexible combination of data movements.

Both postprocessing and in situ.

Scales well.

To Do

Finish installing existing code

AMR and unstructured decomposition

Particle decomposition

Hybrid parallelism?

Limitations

Low level data type

Intrusive in situ

  Takes space, can crash, requires recompilation

Requires effort on the part of the user

  Needs a program and programmer.

Data Movement Support for Analysis

https://svn.mcs.anl.gov/repos/diy/trunk

Peterka et al., Scalable Parallel Building Blocks for Custom Data Analysis, to appear in LDAV '11

Tom Peterka

tpeterka@mcs.anl.gov

Mathematics and Computer Science Division

CScADS Summer Workshop
7/26/11